504-REN-UI-ImplG

MISSION OPERATIONS AND DATA SYSTEMS DIRECTORATE

Renaissance User Interface Implementation Guide

August 1994

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

Renaissance User Interface Implementation Guide

Prepared for

GODDARD SPACE FLIGHT CENTER

By

COMPUTER SCIENCES CORPORATION

Under

Contract No. NAS5-31500

August 1994

Reviewed by:		Approved by:	
XXXX	Date	XXXX	Date
XXXX	Date	XXXX	Date
		XXXX	Date

Abstract

This *Renaissance User Interface Implementation Guide* provides a framework that encourages consistency among user interfaces being enhanced or developed for use within the Renaissance project within the Mission Operations and Data Systems Directorate. The guide is intended primarily for system developers who need guidelines on developing user interface software that will be both intuitive to the user and easy to integrate with the other software building blocks used within the Renaissance software architecture. A secondary audience is Renaissance systems users who can gain information on the underlying user interface design principles inherent in the systems that they will be operating.

Preface

This document represents the preliminary version of the *Renaissance User Interface Implementation Guide*. This version has focused on preparation of material necessary to define the standards and guidelines for the Renaissance user interface building blocks that will be used to support the Advanced Composition Explorer mission. Therefore, the standards and guidelines found in Section 2 can be viewed as complete. The later sections of the document were deemed less critical to this initial stage of Renaissance and are incomplete.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
All	Original

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
	Original	August 1994	

DCN Control Sheet

DCN Number	Date/Time Group	Month/ Year	Section(s) Affected	Initials

Contents

Section 1. Introduction

Section 2. User Interface Standards and Guidelines

Section 3. Methodology for User Interface Development

Section 4. Style Principles

Section 5. Renaissance Controls, Groups, and Models Reference Pages

Appendix. Certification Checklist

Abbreviations and Acronyms

References

Figures

Section 1. Introduction

1.1 Purpose and Scope

This *Renaissance User Interface Implementation Guide* provides a framework that encourages consistency among user interfaces being enhanced or developed for use within the Renaissance project within the Mission Operations and Data Systems Directorate (MO&DSD). The guide is intended primarily for system developers who need guidelines on developing user interface software that will be both intuitive to the user and easy to integrate with the other software building blocks used within the Renaissance software architecture. A secondary audience is Renaissance systems users who can gain information on the underlying user interface design principles inherent in the systems that they will be operating.

This document is maintained by the User Services Working Group (USWG), which reports to the MO&DSD Renaissance Team. The working group will update the document to reflect any changes made to the selections of user interface technology or principles for Renaissance.

1.2 Document Organization

This document is divided into five sections and one appendix. The first section provides background information on the purpose and organization of the document. Section 2 describes each of the standards and guidelines that have been identified to ensure that the goals set for the Renaissance user interface are being met. Section 3 describes the recommended process for developing user interface software targeted for a Renaissance system. Section 4 defines the user interface style principles that govern the "look and feel" of Renaissance user interfaces. This section shows how the basic *Open Software Foundation (OSF)/Motif Style Guide* (Reference 1) look and feel has been augmented for the Renaissance project. Section 5 provides detailed information on the new components available to Renaissance user interface developers that are in addition to the standard set supplied with Motif. Finally, the appendix contains a certification checklist that can be used in conjunction with the OSF/Motif checklist to verify the compliance of a user interface with the principles spelled out in this document.

1.3 Related Documents

The following documents provide additional information about Renaissance, the USWG, or OSF/Motif:

- Open Software Foundation (OSF)/Motif Style Guide
- Renaissance Standards
- Renaissance User Services Transition Plan
- Renaissance User Services Building Block Specifications

1.4 Terminology

The following terms are defined in this section and appear throughout the document:

Application: A software element external to the sphere of the USWG for which the USWG must provide a user interface. Although many applications will fall into the sphere of the Applications Services Working Group, this definition permits elements from the data services, simulation, and spacecraft communication working groups to be considered applications. This term is most commonly used in this document to refer to the non-user interface related portion of a software function.

Building block: A loose term for a generic or tailored subsystem, element, or subelement.

Component: An element within a graphical user interface (GUI) that is identifiable to the user as a distinct entity. Examples of components include pushbuttons, menu bars, and text fields.

Generic: A generic system, subsystem, element, or subelement is one that is always used with no modifications from mission to mission. Mission configurability may be achieved through data parameters or through tailoring.

Guideline: A characteristic of a software building block that can be evaluated to determine the degree of applicability that the building block possesses for use within the Renaissance environment. Guidelines will be used both to steer the development of new building blocks and to evaluate the legacy building blocks available within MO&DSD to determine which should become the baseline set of Renaissance building blocks.

Mission-specific: A mission-specific system, subsystem, element, or subelement is one that is developed and managed for a specific mission.

Software backplane: The set of Renaissance-standard applications programming interfaces (APIs) for providing communications between software elements. Conceptually, elements "plug into" the standard interfaces of the backplane (in analogy to a hardware backplane).

Software element: An independently executing software unit that communicates with other software elements solely through the standard interfaces defined by the Renaissance software backplane. A software element is constructed from subelements that may be linked together or may be separate tasks or processes. Subelements within a software element may communicate among themselves by means other than the backplane.

Software subelement: A configuration-controlled component of one or more software elements. Subelements may themselves have structure, but the subelement level is the lowest level of concern for configuration of a system for a particular mission.

Standard: A set of principles, conventions, and/or programming interfaces established by either the computer industry or the Renaissance team to govern the development of software. The X Window System, OSF/Motif, and Common Desktop Environment (CDE) are all examples of industry standards.

Subsystem: A standard decomposition of a system, encompassing those software elements necessary to carry out a specific high-level functional area. For example, Spacecraft Operations Monitoring and Mission Planning might be subsystems (these examples are for illustrative purposes only and do not imply a definitive decomposition of Renaissance systems).

System: A complete operational environment with clear boundaries and well-defined external interfaces crossing these boundaries, such as the Mission Operations Center, Science Operations Center, and Spacecraft Integration and Test systems.

Tailored: A tailored system, subsystem, element or subelement is one with some mission-specific code added to a generic part in some standard way (e.g., linked in via well-defined hooks).

User interface: A software element that falls within the sphere of the USWG. Thus, from the point-of-view of the USWG, all software elements are either applications or user interfaces.

Widget: A software subelement that conforms to the X toolkit and Motif widget library standards for implementing user interface components. Examples of widgets are XmPushButton and XmTextField. Custom widgets created for the National Aeronautics and Space Administration (NASA) environment are also discussed in this document.

Section 2. User Interface Standards and Guidelines

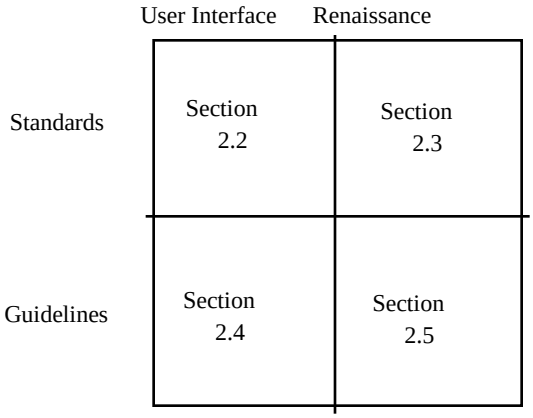
2.1 Renaissance User Interface Goals

Effectively merging the ground support system functionality traditionally provided by MO&DSD into an integrated system requires that the user perceive the system as a single logical entity. A critical factor in this perception is the consistency between the user interface elements presented to the user. Achieving consistency is the foremost goal for the development of Renaissance user interfaces and also the primary intent of this document. Several secondary goals have also been defined. Renaissance user interface goals are as follows:

- Maintain consistent look and feel among user interface elements
- Improve ease of use over user interfaces found in MO&DSD legacy systems
- Empower the users, giving them control over the applications they operate
- Maintain compliance with industry and open systems standards
- Reduce effort required to develop, enhance, and maintain new user interfaces
- Involve users in the development of user interfaces

The standards and guidelines that are steering the Renaissance user interface development have been selected with these goals in mind and are presented in detail in this section. Adhering to these principles should ensure that the user interfaces used by Renaissance will converge into a consistent, natural, and powerful environment for users of spacecraft ground data systems.

The standards and guidelines have been divided into four sections. The sections have been chosen to each represent a single quadrant on the following diagram:



2.2 User Interface Standards

This section describes the user interface standards that have been chosen for Renaissance. The first three subsections describe computer industry standards representing the most prevalent user

interface technology on UNIX workstations at this time. The fourth standard, Renaissance style principles, is a merger of the industry style conventions detailed in Reference 1 and the specific Renaissance conventions described in Section 4 of this document.

Each subsection briefly describes the item and then lists the key attributes that a Renaissance user interface building block needs to meet the intent of this standard.

2.2.1 X Window System, Version 11

The X Window System is the industry-standard window environment for a network of cooperating workstations, personal computers (PCs), and X-terminals. First released by the Massachusetts Institute of Technology (MIT) in the late 1980s, X has blossomed into the core technology that makes GUIs possible on a network driven by UNIX workstations. Because UNIX workstations have been designated the platform of choice for Renaissance, the X Window System is the natural choice for a windowing system.

X Window System Release 5 (X11R5) is widely in use at this time, and products built on this version of the software are available from most vendors. This release is also the version of X that has been chosen for CDE. X Window System Release 6 is now available from MIT, but has not yet been included in software releases from major vendors.

Key Attributes: Renaissance user interface building blocks shall be based on X11R5 or greater.

2.2.2 Open Software Foundation Motif

The OSF/Motif window manager and widget library are software products layered on top of the X Window System to provide a standard look and feel to a GUI. Motif was first released in 1989 and has become the toolkit of choice on UNIX workstations over its rival, OpenLook from Sun Microsystems. The Motif style is also well documented in the OSF/Motif Style Guide, which enforces a consistent presentation of information and style of interaction between a graphical application and its user. Because Motif is the standard look and feel for the UNIX environment, it is the natural choice for Renaissance.

Motif Release 1.2, based on X11R5, is widely supported by UNIX vendors at this time. Release 1.2.3 has been chosen as the initial version for CDE. Release 1.2.4 has also recently come available.

Key Attributes: Renaissance user interface building blocks shall be based on Motif Release 1.2.3 or greater.

2.2.3 Common Desktop Environment

CDE is a attempt by UNIX workstation vendors to create a standard user environment such as those existing for Macintosh users across all Macintosh PCs or with the Windows environment for PC users. Announced in late 1993 by the Common Open Software Environment consortium of vendors, CDE builds on the most prevalent user interface technology available for UNIX systems at this time. X11R5 and Motif Release 1.2.3 are the most important components of CDE. Other user interface technologies included in CDE are the Hewlett-Packard (HP) Visual User Environment, which provides a graphical interface to the UNIX desktop; an expanded version of

the OSF/Motif Style Guide called the *CDE User Interface Style Guide*; and the X Window *Inter-Client Communications Conventions Manual (ICCCM),* which specifies how X applications should communicate with each other.

CDE development toolkits should be available to UNIX vendors in 1994. Actual releases of this suite of technology by the vendors will probably not occur until 1995. Two levels of integration with CDE have been specified: basic and full. Renaissance user interface building blocks will need to work to become as closely integrated with CDE as possible to form a consistent environment for users of Renaissance systems.

Key Attributes: Renaissance user interface building blocks shall support CDE basic integration by the end of calendar year 1995. The requirements for full integration with CDE are TBS.

2.2.4 Renaissance Style Principles

The Renaissance user environment is based on the look and feel for GUIs described in Reference 1. This industry document will guide the development of all Renaissance user interfaces. The Renaissance project will document any additions or subtractions to Motif's list of style requirements within Section 4 of this document. The appendix provides a checklist that itemizes each new Renaissance requirement and each Renaissance change to the standard Motif rules. This checklist can be combined with the checklist in the OSF/Motif Style Guide to completely analyze the conformance of a building block with the Renaissance style principles.

The OSF/Motif Style Guide has been available since Motif’s introduction 1989. It is updated with each major Motif release (current version is 1.2) and will be superseded by the *CDE User Interface Style Guide* within the next few years.

Key Attributes: Renaissance user interface building blocks shall be compliant with the OSF/Motif Style Guide. The only exceptions to this rule are exemptions from normal style guide principles that have been listed in Section 4 or the appendix of this document. In addition, building blocks shall be fully compliant with the Renaissance-specific requirements of Section 4 and the appendix by the end of calendar year 1995.

2.3 Renaissance Standards

This section describes the Renaissance projectwide standards that apply to work within the user services. The complete set of Renaissance standards is documented in *Renaissance Standards* (Reference 2). The first two subsections describe computer industry standards reflecting the most common technologies for developing a reusable, portable software base on UNIX workstations at this time. The third item, Renaissance development standards, is included to ensure that a common approach is taken for developing and maintaining Renaissance software components.

Each subsection briefly describes the item and then lists the key attributes that a Renaissance user interface building block needs to meet the intent of this standard.

2.3.1 Implementation Language

The architecture working group is specifying Renaissance-wide standards for development, including a list of acceptable implementation languages. These languages have been chosen

based on criteria such as breadth of use, portability, features, and support for object-oriented programming.

Renaissance currently allows new development in C, C++, and Ada. The latter two object-oriented languages are preferable because the long-term software architecture for Renaissance is anticipated to be based on an object broker model. Legacy code written in FORTRAN may also be included in a Renaissance building block, though no new FORTRAN development is anticipated. Also note that the X Window System toolkit requires that custom widgets be developed in C.

Key Attributes: Renaissance user interface building blocks shall be coded in C or C++ if possible for ease of integration with the X Window System and Motif. C is preferred for custom widgets; C++ is the language of choice for a new software part.

2.3.2 Portable Operating System Interface for UNIX

The portable operating system interface for UNIX (POSIX) standards have been developed by the computer industry to promote portability of software across hardware platforms developed by different vendors. In particular, the POSIX 1003.1 standard defines an API for software that requires system calls for process management, device input/output (I/O), and file system I/O. Likewise, the draft POSIX 1003.4 standard defines an API for real-time extensions to the base 1003.1 standard. Renaissance software building blocks should make POSIX calls whenever possible to ensure platform independence.

Key Attributes: Wherever possible, system calls used by Renaissance user interface building blocks shall conform to the POSIX 1003.1 standard or the 1003.4 draft standard for real-time extensions. It is recognized that POSIX may only define a subset of the full functionality required by some Renaissance software parts. For those cases, where additional system calls are absolutely essential, this less portable code shall be isolated in separate files or libraries and documented accordingly.

2.3.3 Renaissance Development Standards

The Renaissance Architecture/Communications Working Group (ACWG) is developing specifications to define the standard mechanisms for developing and maintaining Renaissance building blocks. An example of a development standard would be a tool to allow software to be built across multiple hardware platforms (e.g., Imake). A standard selection of such a tool would allow the entire Renaissance reusable software library to be installed on a new platform in a single operation. The Renaissance development standards are TBS at this time.

Key Attributes: Renaissance user interface building blocks shall be developed according to the implementation standards defined in Section 3.4 of Reference 2.

2.4 User Interface Guidelines

This section describes the user interface guidelines that have been chosen for Renaissance. The first three guidelines have been chosen to evaluate the consistency and ease of use of a building block. The final three items are included to reflect the ease of maintenance and reusability of the

user interface software by analyzing the decoupling of the software from any specific application, layout, or object behavior.

Each subsection briefly describes the item and then lists the key attributes that a Renaissance user interface building block needs to meet the intent of this guideline.

2.4.1 Data Import/Export to Other Applications and Devices

The Renaissance user environment needs to operate as a single, integrated entity even though the software building blocks within that environment have been developed by different organizations and each have a unique heritage. A key element of any integrated user environment is the ability to share data among tools and devices within that environment. Thus, Renaissance user interface building blocks need to support a certain level of data interchange.

With Motif Release 1.2, a standard was released for "drag and drop" behavior within a Motif application or between Motif applications. Drag and drop is one of four approved Motif techniques for transferring data in a Motif environment. The other notable mechanism of these four is transfer of data through the clipboard. Because Renaissance user interfaces must conform to the OSF/Motif Style Guide, any data transfer between or within building blocks should use one of the style guide-approved techniques. Data import and export also should be supported to files and printers.

Key Attributes: Renaissance user interface building blocks shall support the following data import and export operations where appropriate:

- Print data to a Postscript laser printer
- Import data from other user interface elements through a Motif-approved transfer technique
- Export data to other user interface elements through a Motif-approved transfer technique

2.4.2 Scripting (Macro) Capabilities

The Renaissance operations concept calls for a minimal staff of Mission Operations Team personnel, with the reduction in size made possible in part by a high degree of automation of operational activities. This concept, translated into the realm of user interface building blocks, means that all user interfaces should have a method for scripting user operations to allow for automation of frequently repeated tasks. This automation can be achieved within a user services building block in any of the following ways (in order of most to least complex to the user):

- Scripting language that allows preprogramming of operational procedures and alternate flows of control through the scripts
- Macro definition capability that allows multiple user operations to be chained together into a single operation
- Recording mechanism that automatically captures user activity and records the sequence for later playback

Also note that Renaissance user activity may be scripted across software building blocks. The standard mechanism for sequencing operations will be a shell script. Because an updated version of the Korn shell is part of the CDE specification, Korn shell scripts are the recommended approach for sequencing operations between building blocks. Note that in most cases, these scripts can be developed by mission development teams and do not need to be explicitly delivered with the software for a particular building block.

Key Attributes: Renaissance user interface building blocks shall internally allow the user to automate repetitive user tasks through a scripting capability. This requirement may be satisfied through a scripting language, a macro definition capability, or an equivalent functionality. Furthermore, building blocks shall provide the capability to be invoked with a single UNIX command or script invocation so that building block operations can be automated with shell scripts.

2.4.3 Customizable by User

GUIs have significantly increased the ease of use of computer systems. Most early graphical interfaces imposed a fixed layout of menu options with built-in fonts and colors. However, modern user interface software is expected to be easily customized by the user. Color, font, and icon and menu selection is left up to a user's discretion in most of today's PC-based commercial applications.

Although the Renaissance user environment does not have quite as stringent a set of requirements for personalizing user environments as today's commercial software, some basic level of capability in this area is mandatory. Most important is that different projects or installations of a user interface building block can tailor stylistic elements of a user interface to their taste. In some applications, projects will also require the ability to design their own user interface panels, as well. Customization on a per-user basis is less important because most Renaissance workstations will be shared among several users during the course of a given day, perhaps without a new user session being started.

Key Attributes: Renaissance user interface building blocks shall allow the user to tailor certain attributes of a user interface according to personal preference, for example, placement of entries on menus, color assignments, and fonts. The building blocks also shall allow user definition of new display windows as appropriate to the application.

2.4.4 Decoupled From Underlying Application

The Renaissance approach to system development has partitioned ground data systems into service areas such as user services, data services, and application services. To facilitate this functional division, as well as meet the traditional software design principle that the presentation layer of a software product should be separate from the underlying application, Renaissance user interface building blocks should only be loosely coupled with the application for which they provide the human-machine interface. Keeping the user interface software separate from the underlying application should allow the user interface building block to evolve to meet the standards, guidelines, and style principles found in this document without any impact to the application it supports.

Key Attributes: The software for Renaissance user interface building blocks shall be maintained separately from the code for the underlying application, and the two shall execute as separate software processes. These two elements that compose a complete application shall communicate as defined in the specification for the Renaissance software backplane (Reference 3).

2.4.5 Separation of User Interface Layout and Code

User interface design is often most successful when the developer and user work closely in an iterative manner to refine the user interface, especially in terms of display layout and presentation. Therefore, the portions of a user interface that are most likely to change are the screen layouts, colors, and fonts—all aspects of a user interface's look and feel, not the underlying application code.

Changes to colors and fonts were discussed in Section 2.4.4. However, the issue of ease of maintenance for a user interface's layout remains. Several commercial tools and standards exist to facilitate maintenance of user interface layouts. The User Interface Language (UIL) provided with Motif is becoming a defacto standard for representing user interface layouts. Furthermore, tools such as Builder Xcessory and UIM/X allow interface layouts to be maintained with a graphical layout tool instead of in a formal language. Support for UIL layouts and integration with tools of this nature is strongly encouraged for Renaissance user interface building blocks.

Key Attributes: Renaissance user interface building blocks shall allow user interface layouts to be maintained independently of the underlying user interface and application software. Changes to layouts shall not require corresponding code modifications, recompilation, and/or relinking of the building block software.

2.4.6 Provides Custom Widgets

Although the standard Motif widget set contains most of the basic components needed to create a GUI, additional objects are required for the satellite control and real-time display monitoring environment. Several projects within MO&DSD have already created custom widgets based on the class structure provided with the Motif widget set. Because these software components have already been written within a highly stylized object-oriented framework, they can be readily shared among user interface building blocks.

The USWG has therefore decided that a Renaissance widget library should be initiated to pool these new display objects into a single software library. Once formed, this library will be augmented as new widgets are needed by a building block. Furthermore, this library will be enhanced so that consistency is achieved between widgets developed by different organizations within MO&DSD. The ability to accommodate widgets from this library and commercial sources should be included in Renaissance user interface building blocks, where appropriate.

Key Attributes: Any custom widgets developed for Renaissance user interface building blocks that are of general utility shall be contributed to the Renaissance widget library for use by other building blocks. These widgets shall be implemented and documented according to the standards for Renaissance widgets included in Section 3 of this document, which includes complete information on how to resolve make/buy/build/reuse decisions considering use of widgets in user interface building blocks.

2.5 Renaissance Guidelines

This section describes the Renaissance projectwide guidelines that apply to work within the user services. These 10 principles reflect the compatibility of a component with the Renaissance software environment, the maturity of the code, supporting documentation, and support facilities, and the degree of portability across platforms and missions.

Each subsection briefly describes the item and then lists the key attributes that a Renaissance user interface building block needs to meet the intent of this guideline.

2.5.1 Platforms Supported

The Renaissance ACWG is currently defining the hardware architecture for near-term Renaissance missions. Although every attempt will be made to standardize the hardware platforms both within a single ground data system and across missions, Renaissance software building blocks will need to be ported to new hardware platforms over time, at least to take advantage of technological advances within a single vendor's product line.

One way to achieve software portability is to routinely maintain software components on more than one platform. Thus, Renaissance building blocks should be routinely built and tested on at least two reference platforms. The initial reference platform for Renaissance is the HP workstations to be used for the Advanced Composition Explorer (ACE).

Key Attributes: Renaissance user interface building blocks shall be able to operate on any UNIX workstation platform supporting the X Window System, Motif, and POSIX system calls. It is acknowledged that a software port may be required for certain platforms as defined in Section 2.5.8. Building blocks shall run on the current reference platform for Renaissance—HP workstations running HP/UX 9.0 or greater.

2.5.2 Commercial Off-the-Shelf Usage

An underlying principle of open systems development and the Renaissance architecture for ground data systems is that commercial off-the-shelf (COTS) software products should be used whenever possible to provide cost-effective systems. However, the selection of COTS software should be a formal tradeoff study process that examines the relative merits of make or buy in each situation. Furthermore, the use of a COTS package should not undermine the standards and guidelines documented in this section. For example, a COTS package without a Motif style interface should not usually be considered acceptable for a Renaissance system.

Key Attributes: Renaissance user interface building blocks shall use embedded COTS software if tradeoff studies determine that the life-cycle cost is reduced through the use of such a COTS product and if the product itself conforms to the standards and guidelines presented in this section.

2.5.3 Renaissance Interface Compatibility

The Renaissance ACWG is developing specifications defining the standard mechanisms for interprocess communication between Renaissance building blocks. This set of interface definitions is known as the Renaissance software backplane. Standard protocols for items such as

requesting and transmitting data, issuing control messages to another process, and broadcasting system event messages will be included in the backplane definition.

Because most user interface building blocks were developed prior to the initiation of the Renaissance project, they currently use interprocess communication techniques that differ from the Renaissance software backplane. Each building block will need to transition to this new set of interfaces. An initial definition of the Renaissance software backplane will be completed in mid 1994. It is anticipated that this backplane will evolve over time to remain current with technological advances and emerging standards such as the Distributed Computing Environment from OSF or the Common Object Request Broker Architecture.

Key Attributes: Renaissance user interface building blocks shall communicate with other Renaissance software parts using the protocols and mechanisms defined as part of the Renaissance software backplane (Reference 3).

2.5.4 Documentation

All mature software products are shipped with high-quality documentation describing how to configure and use the product. Renaissance building blocks should adhere to this same standard. The documentation should be sufficient to allow the building block to be easily configured for a new mission. Furthermore, end-user information on running the software shall be included. This end-user documentation should be available as online help as well (Section 2.5.6).

The initial documentation efforts will aim at ensuring that a complete set of information is available on each building block. Over time, the documentation for each building block should evolve toward a standard presentation. This consistent format will allow users of multiple building blocks to easily find the information they require because it will be presented with a common approach for all Renaissance components.

Key Attributes: Renaissance user interface building blocks shall include a complete set of documentation for both end users of the building block and system integrators configuring the building block for a particular mission. Documentation standards are listed in Reference 2.

2.5.5 Maturity

Renaissance building blocks represent software components that will be used for operational support of a series of NASA missions. These components need to be tested, proven software that will work reliably in an operational setting. A process should be defined in which these components are formally tested and accepted. Although some of the Renaissance building blocks are just in the prototype stage of development, it is required that they all be mature products before the launch of the initial Renaissance missions.

Key Attributes: Renaissance user interface building blocks shall be mature software products already in use within operational systems whenever possible. A formal acceptance test process shall be in place for all releases of building block software.

2.5.6 Online Help Facility

In the Renaissance environment, a single set of Mission Operations Center users is presented with dozens of software components, each requiring detailed knowledge to properly execute. Maintaining all of this information in hardcopy format will be cumbersome and will require the user to search multiple volumes for a key piece of information. A better approach is to move end-user documentation into an online format.

None of the user interface building blocks selected for Renaissance have currently implemented an online help capability. This shortcoming actually affords Renaissance a unique opportunity to create a consistent online help facility from the ground up. The CDE comes with a built-in help system that will be used as the basis for help on Renaissance building blocks.

Key Attributes: Renaissance user interface building blocks shall support an online help capability to reduce the need to rely on printed documentation. The CDE help facility, when available, shall be used as a baseline library for implementing help functionality. Support for context-sensitive help and hypertext links between help windows is highly desirable.

2.5.7 Performance

All well-designed user interfaces provide immediate feedback to the user to indicate that the user action has been recognized and is being acted on. Renaissance user interface building blocks also must conform to this design principle. If operations are going to take a significant amount of time to complete, the user should have a visible indication of the fact that the operation is in progress. Use of slider bars to denote percent complete is one helpful way of conveying information on work in progress to the user. Actions that might lock out the user for a significant period of time should allow the user to cancel the activity.

Key Attributes: Renaissance user interface building blocks shall provide feedback to the user for any action that does not complete within a few seconds. The user should be able to cancel any operation that takes more than 30 seconds to complete. Use of dialog boxes denoting that work is in progress is highly desirable.

2.5.8 Portability

In addition to following the criteria listed in Section 2.5.1, software portability should be ensured by following the commercial software standards listed previously (e.g., X11R5, Motif, POSIX) and by attempting software ports to new platforms. The effort required to port a building block to a new platform should be minimal if the software is properly written for platform independence.

Key Attributes: Renaissance user interface building blocks shall be portable to a new POSIX platform (Section 2.5.1) in less than 1 month.

2.5.9 Availability of Support

Commercial software products typically are marketed with an option for on-call software support from the vendor. Renaissance building blocks should adhere to this same standard. The organizations that maintain Renaissance software building blocks, known as centers of expertise (COEs), shall be responsible for providing this support service. This support will supplement the

information for system integrators discussed in Section 2.5.4. COEs will be expected to have a dedicated staff supporting each building block that shall be available to answer questions concerning use of that software.

Key Attributes: Renaissance user interface building blocks shall be actively maintained by a COE within MO&DSD. A support office shall be maintained for each building block by the COE. The personnel who provide this support office capability shall be available to assist projects using the software during normal working hours.

2.5.10 Verbatim Reuse From Mission to Mission

Several projects within MO&DSD over the last 10 years have shown that high levels of software reuse are achievable and can lead to drastically reduced mission support costs. Following this heritage, Renaissance has established high levels of mission-to-mission software reuse as a key strategy. This reuse is accomplished through the software building block concept.

Each building block shall be configurable through data files or startup parameters to as high a degree as possible. Mission-specific additions to building blocks will be allowed only if it is clearly not cost effective to add functionality for a mission in a reusable fashion. Configuration control will be maintained so that these mission enhancements are isolated from the core set of reusable software.

Key Attributes: Renaissance user interface building blocks shall be usable on multiple missions without code changes to the core generic software for that building block. Mission-specific enhancements shall be permitted, but only by inserting code into well-defined hooks that are maintained separately from the core software for that building block.

Section 3. Methodology for User Interface Development

3.1 Software Reuse Process

This section describes the first step in developing a new user interface for Renaissance, namely, reviewing the existing Renaissance software library to determine which components can be reused instead of built from scratch. This process entails four steps as detailed in the following subsections. Once this reuse analysis is complete, development of a new user interface component (Section 3.2) and/or a new widget (Section 3.3) can commence as required.

3.1.1 Review Existing Building Blocks

This subsection is TBS.

3.1.2 Itemize New Capabilities

This subsection is TBS.

3.1.3 Determine Scope of Effort

This subsection is TBS.

3.1.4 Initiate Development

This subsection is TBS.

3.2 User Interface Development Process

The user interface represents the subsystem that channels communications between system users and system applications. This section describes the processes that developers working with users employ to create this subsystem and to adapt it to a particular mission and application. Because the user interface subsystem is the user's direct point of contact with the system, it is essential that it clearly reflect the user’s needs and desires. The criteria for a development methodology include the following:

- **Reuse** of existing components
- **Consistency** of user interfaces
- **Adaptation** to various applications
- Expectation of **change**
- **Quick demonstration** of proposed user interfaces
- **Relationship** with other system components (e.g., applications)

- **Close interaction** with users to gather their needs and desires

The key terms of these criteria are highlighted. The development methods described in the following subsections address each of these criteria.

A typical assembly line or waterfall development model would not meet these criteria. Renaissance will use an object orientation (Reference 4) and spiral development methodologies (Reference 5) for user interface development combined with joint application design (JAD) teams. Spiral development minimizes risk by providing tangible, but changeable interim products through each development cycle. (Early interim products are also known as prototypes.) Joint application development ensures that the system under development adequately reflects user needs and perceptions. Both aspects support development of highly interactive systems. They prevent most errors or misperceptions in the requirements generation and system development process. They allow for early detection and correction of those that do get through.

3.2.1 Object-Oriented Concepts

An object-oriented specification method readily supports creation of a consistent set of user interfaces through the concepts of inheritance and abstraction. Object orientation employs the concepts of objects and classes. An object is a specific encapsulated unit of methods and attributes that supports an application. Objects communicate exclusively by messages. Objects are specific instances object-oriented class.

3.2.1.1 Abstraction

Object-oriented abstraction is the process of identifying, specifying, and encoding common aspects of a class of objects. Thus, a class is a template for the creation of objects with identical methods and attributes. As users perform their tasks, the system often creates instances of a class. This dynamic creation process is also know as instantiation. Each of these objects will be the same except for the values of its attributes. Thus, abstraction supports both **reuse** and **consistency**.

For example, a designer can specify and encode a one-button dialog box class. This dialog box can be instantiated a message “Network Printout Ready” with a blue background, or as an error message “File Not Found” with a red background. In these cases, the message and the background colors are attributes.

3.2.1.2 Inheritance

Object-oriented inheritance provides a mechanism for designers to efficiently specify a set of related classes. Using inheritance, a designer can state that a new class **inherits** all of the characteristics of the parent class with exceptions, which allows the designer to specify and build each class of objects in terms of other classes. Thus, inheritance also supports **reuse** and **consistency** while allowing **adaptation** to various applications. For example, the designer can specify a two-button dialog box class. The designer specifies that the two-button dialog class inherits from the one-button class, except that it will have an additional button.

3.2.2 Spiral Development

A two-dimensional polar coordinate diagram (Figure 3–1) depicts the spiral development method. The radius is system maturity. The angle represents development activities. The process traces a spiral path that begins with a raw concept at the origin and proceeds outward through iterations of development activities, each time reaching higher levels of maturity.

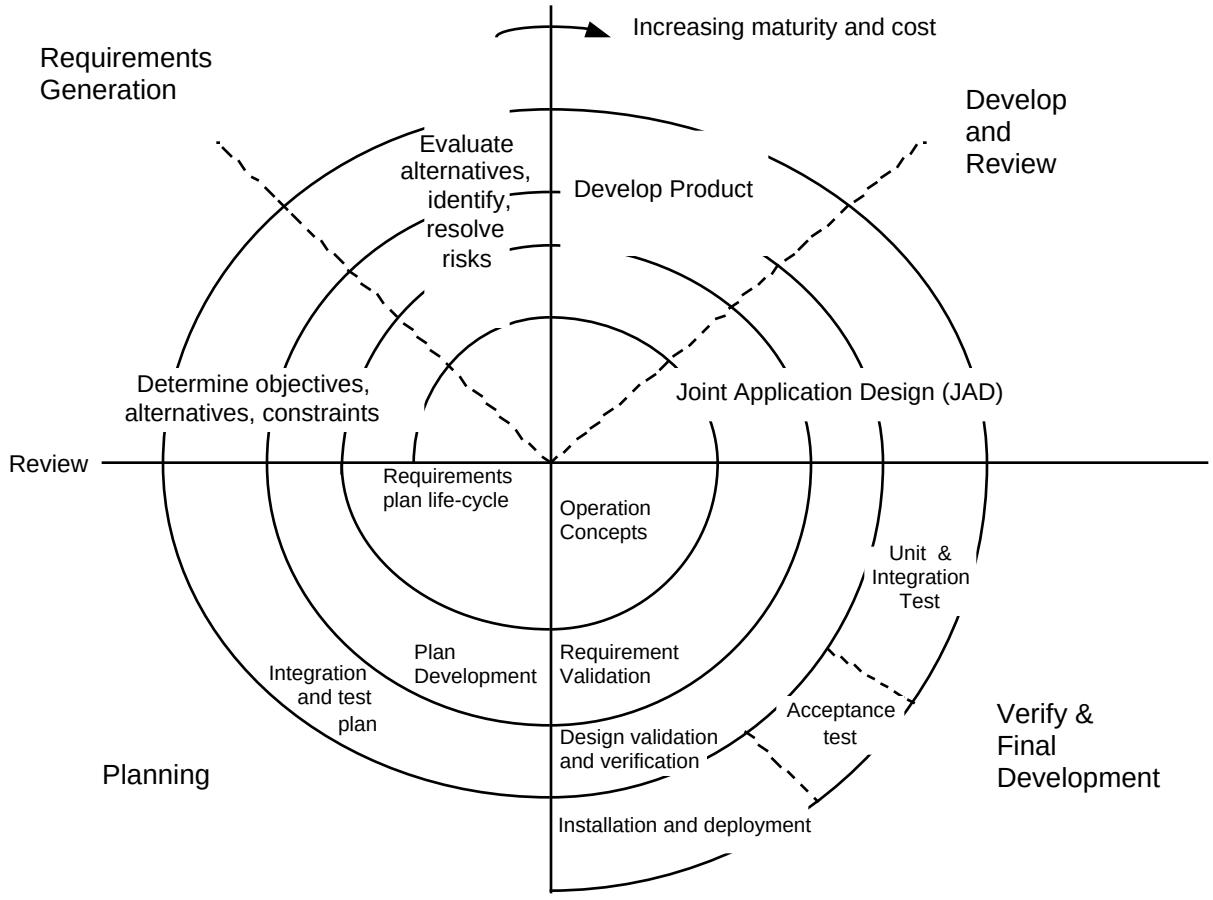


Figure 3–1. Spiral Development Methodology

Effective project development requires several iterative stages of planning, requirement generation, evaluation, and development. The spiral development method accommodates this requirement through recursive design. The model in Figure 3–1 depicts four cycles, which represent a typical number of cycles. This process allows and even expects **change** as the system matures.

Each cycle begins with planning. In this initial phase, developers plan the expected number of iterations along with milestones and activities for each cycle. Each cycle lasts between 3 and 4 months; the overall project will last between 12 and 16 months.

The next phase is requirements generation. During the initial iteration, analysts focus on a vision and strategy for the proposed system. They translate these into general system needs. In this phase, analysts also assign risks and benefits to potential development techniques and choose the most promising of these risks and benefits.

Developers use advanced tools to construct the user interface. These tools permit **quick demonstrations** of proposed user interfaces, which in turn become the de-facto requirements. The user interface is only a two-dimensional Hollywood set without communications with the rest of the system infrastructure. To add depth, user interface developers working with application developers establish message protocols and data structures to enable communications. These messages specify the **relationship** with the other applications within the system. Initially, developers can model these interfaces using drivers, stubs, dialog boxes, and simulators. As the overall system matures, developers integrate the user interface with other subsystems. Figure 3–2 illustrates the role of the user Interface within the overall system.

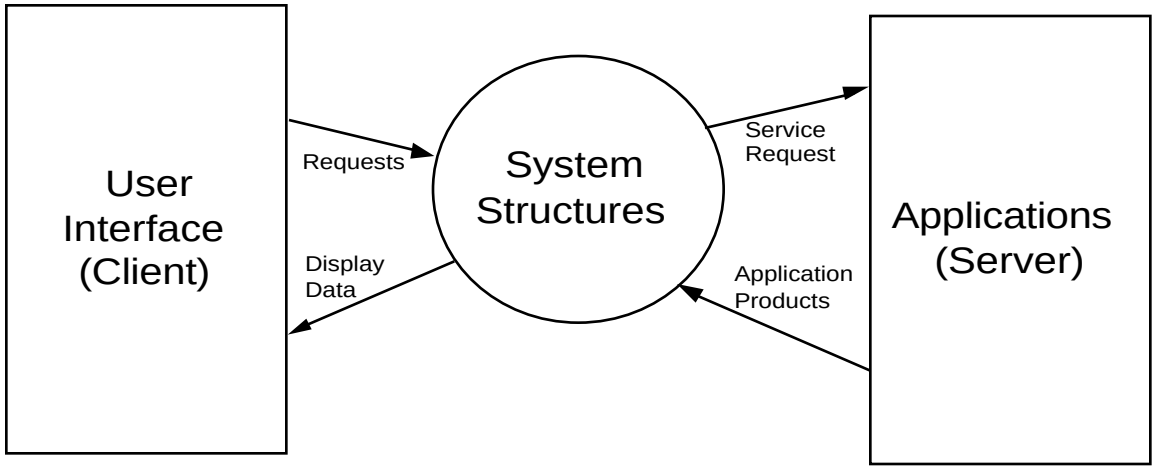


Figure 3–2. User Interface System Environment

Developers then demonstrate the interim products to analysts and users during a JAD session (Section 3.2.3). During demonstrations, they solicit and record user reactions and assessments. The data from the demonstrations is then used to plan the next steps in the development process. In later cycles, as the system under development begins to take shape through construction and demonstration, analysts add more detail to the requirements. A key concept is that each build is an actual, although incomplete, product. This means that there is no throwaway development; each cycle and phase contributes to the final product.

During the final cycle of the development process, methods and products become more rigorous. Developers use configuration control procedures and tools to maintain system integrity and control. They integrate and test the user interface with the overall system. Finally, the testers and users perform acceptance testing.

3.2.3 Joint Application Design

JAD is a group activity in which participants help develop an application. Using JAD, a facilitator brings users and developers together to specify requirements. The facilitator holds joint meetings with users and developers to review requirements and interim products. Typically, developers present requirements as user interface prototypes, which task participants critique and refine. These prototypes become the actual user interface subsystem as the system matures. The user interface development method described in this document employs JAD only for the evaluation portion of the development process.

3.2.3.1 JAD Participants

The roles within the JAD team are the facilitator, a scribe, users, customers, and developers. A participant may have more than one role in the development process. For example, the facilitator may be a developer. Customers buy the system. Users are the people who operate the system. Optional roles are representatives from the system architecture group and observers. The facilitator understands the development process, needs of the development task, schedule, and roles of the participants. The scribe records meeting minutes, notes, action items, and products. User-participants represent the entire community that will eventually use the system. They have domain knowledge and experience with spacecraft characteristics and ground support missions. They are typically mission analysts, operators, or controllers.

Developers know and understand computer systems. They are familiar with X Window, Motif, and Renaissance user interface standards. They are also familiar with other open-system characteristics, such as software interface standards. They understand the processes of generating requirements, evaluating alternatives, and producing working systems. They are capable of using these tools to develop working prototypes (models) to demonstrate to other task members. In some cases, they may be able to modify the prototype during the meetings. Prototypes serve as dynamic tangible specifications and also become the actual user interface subsystem.

Representatives from the system architecture group understand system and interface standards. They have the authority to modify these standards if necessary. Observers are present to learn about the JAD process and their future roles, e.g., facilitators.

3.2.3.2 JAD Process

The JAD process consists of planning activities, determining objectives, selection, development, JAD sessions, and then reiterating the process.

The JAD session acts as a focus for other steps in the development process. It promotes **close interaction** with users to gather their needs and desires. The JAD session employs rules and procedures that optimize its productivity and foster cooperation among its participants. Other processes in the development cycle help prepare for the JAD session. In addition, the facilitator must prepare or assemble specific materials, including

- Project objectives
- Requirements and criteria
- Renaissance user interface guidelines and standards

- User's tasks (Reference 5, Chapter 12)
- Names and roles of participants
- Interim product (prototype) and equipment for demonstrations
- Agenda
- List of objectives for the JAD session

The facilitator will then prepare and distribute meeting materials and an agenda. The facilitator will arrange meeting rooms, schedules, and tools (e.g., computer, display or projection screen). In some cases, the facilitator may also provide CASE tools, drawing tools, and a GUI prototyper for dynamic modification of the prototype.

JAD sessions begin with a presentation of the JAD tasks, products, and challenges. Developers then present proposed solutions in a format that allows for ample feedback from participants. This process embodies **close interaction** among users and developers.

3.2.3.3 JAD Products

The products of the session are minutes, action lists, and offline assignments. The assignments are resolution of action items, requirements analysis, further development, and presentation of task status and products. Figure 3–3 shows a model of all UI development processes and products.

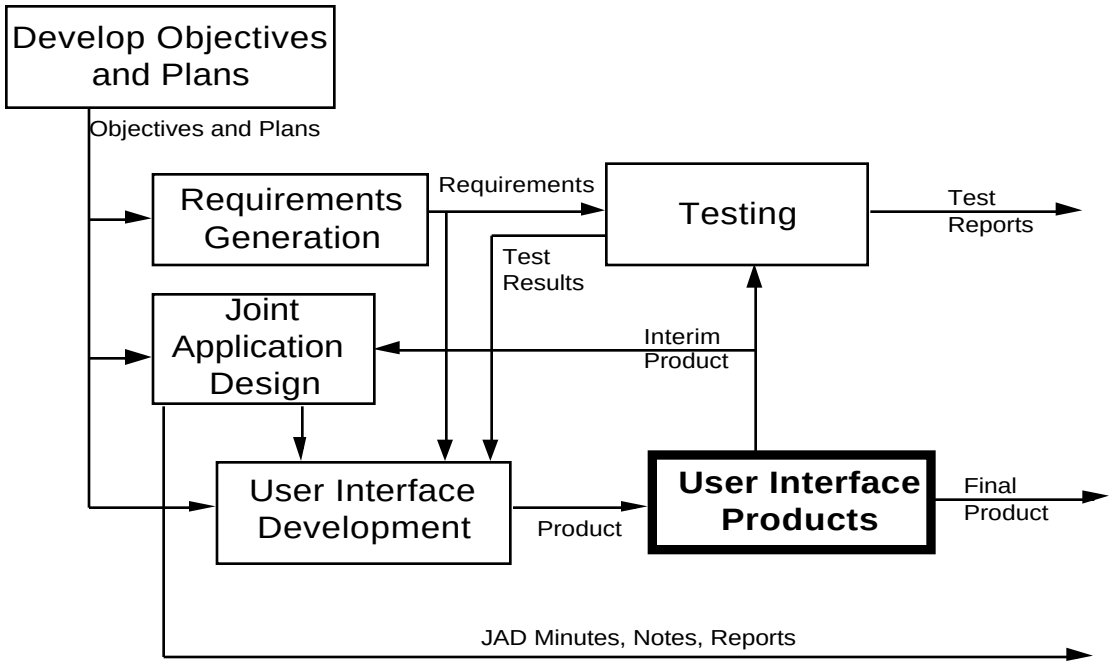


Figure 3–3. Development Products

3.2.4 Completion of the User Interface Process

In the last cycle of the development phase (Figure 3–1), formal products are produced, including

- Design notes
- Code (if any)
- Unit and integration test plans and results
- Acceptance test plans and results
- Configuration control lists
- User interface process

The formal end of the process will normally be the installation of the product and its acceptance by the operational mission team. The process may begin again with the identification of other possible system enhancements.

3.3 Widget Development Process

This section is TBS.

Section 4. Style Principles

The Renaissance user environment is based on the look and feel for GUIs described in Reference 1. This industry document will guide the development of all Renaissance user interfaces. The Renaissance project will be documenting any additions or subtractions to Motif's list of style requirements within this section of the document.

The Renaissance style principles are currently TBS.

Section 5. Renaissance Controls, Groups, and Models Reference Pages

Chapter 9 of the OSF/Motif Style Guide provides detailed information about standard Motif components, user interface models, and concepts in a reference format. Each topic starts on a new page and is organized alphabetically. Typically, one page exists for each component, which usually corresponds directly to a single widget.

This chapter provides the same information in an identical format for each component in the Renaissance widget set. This information is currently TBS.

Appendix. Certification Checklist

The checklist in this appendix itemizes each new Renaissance style principle and each Renaissance change to the standard Motif principles. This checklist can be combined with the checklist in the OSF/Motif Style Guide to completely analyze the conformance of a building block with the Renaissance style principles.

The checklist itself is currently TBS.

Abbreviations and Acronyms

ACE	Advanced Composition Explorer
ACWG	Architecture/Communications Working Group
API	applications programming interface
CDE	Common Desktop Environment
COE	center of expertise
COTS	commercial off-the-shelf
GUI	graphical user interface
HP	Hewlett-Packard
I/O	input/output
JAD	joint application design
MIT	Massachusetts Institute of Technology
MO&DSD	Mission Operations and Data Systems Directorate
NASA	National Aeronautics and Space Administration
OSF	Open Software Foundation
PC	personal computer
POSIX	portable operating system interface for UNIX
UIL	User Interface Language
USWG	User Services Working Group

References

R.1 Cited References

1. Open Software Foundation, *Open Software Foundation (OSF)/Motif Style Guide*, Revision 1.2, Cambridge, MA, 1992
2. Mission Operations and Data Systems Directorate, *Renaissance Standards*, Draft, August 1994
3. —, *ACE Mission Architecture*, Working Draft, July 1994
4. Fayed, M, et al., "Adapting an Object-Oriented Development Method," *IEEE Software*, May 1994
5. Boehm, B., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1998

R.2 Uncited References

Computer Sciences Corporation, *Transportable Payload Operations Control Center (TPOCC) User Interface Specification Methodology*, September 1993

Modell, M., *A Professional's Guide to Systems Analysis*, McGraw-Hill, 1988

Computer Sciences Corporation, *SEAS Systems Development Methodology*, Chapter 8, "User-System Interface Development," December 1993

Pollack, J., *Introduction to Joint Application Design (JAD)*, January 10, 1994

Wood, J., and D. Silver, *Joint Application Design, How to Design Quality System in 40% Less Time*, Wiley, 1989